Jacobian 神经网络算法

甄景贤 (King-Yin Yan)

General.Intelligence@Gmail.com

经典的神经网络 Back Prop 学习算法，它是一个 error-driven 算法，但在很多人工智能的实际应用中，不存在唯一的「理想答案」，而是根据正或负的奖励 (reward) 学习。当答案正确时，奖励 $> 0, \text{error} = 0$；当答案不正确时，奖励 < 0，但 error 仍是不知道的（因为不知道理想答案）。简言之，就是不能用 error-driven 学习。

所以我想出了一个 reward-driven 的学习法：假设神经网络将 $\boldsymbol{x}_0 \mapsto \boldsymbol{y}_0$，它通常也会将 $\boldsymbol{x}_0$ 的邻域 map 到 $\boldsymbol{y}_0$ 的邻域。如果我们想「加强」这个映射，可以将「更大的 $\boldsymbol{x}_0$ 的邻域」映射到「接近 $\boldsymbol{y}_0$ 的邻域」。

这种算法对人工智能应该很重要，暂时我还想不出有什么其他办法，可以做到 [深度] 神经网络的 reward-driven 学习。

将这思想更准确化，可以将 feed-forward 神经网络的构造看成是这样的：

$$\boldsymbol{y} = \boldsymbol{F}(\boldsymbol{x}) \tag{1}$$

$$\boldsymbol{y} = \oslash \overset{L}{W} \ldots \oslash \overset{\ell}{W} \ldots \oslash \overset{1}{W} \boldsymbol{x} \tag{2}$$

其中 W 代表每一层 (layer) ℓ 的矩阵。

F 的**反方向**是：

$$\boldsymbol{x} = \boldsymbol{F}^{-1}(\boldsymbol{y}) \tag{3}$$

$$\boldsymbol{x} = \overset{1}{\gtreqless} \obslash \ldots \overset{\ell}{\gtreqless} \obslash \ldots \overset{L}{\gtreqless} \obslash \boldsymbol{y} \tag{4}$$

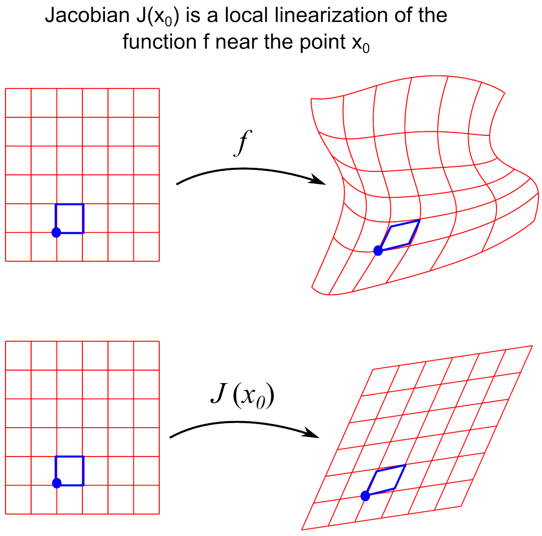
注意：$\gtreqless = W^{-1}$, $\obslash = \oslash^{-1}$，形状不同。

假设在 $\boldsymbol{x}$ 空间有体积元 U，经过 $\boldsymbol{F}$ 变换成 $\boldsymbol{y}$ 空间的体积元 V，那么：

$$U = |J| \cdot V \tag{5}$$

$J = \left[\dfrac{\partial \boldsymbol{F}(x)}{\partial x} \right]$ 叫 Jacobian 矩阵。

在我们的情况下，$|J| = \left|\dfrac{\partial \boldsymbol{F}^{-1}(\boldsymbol{y})}{\partial \boldsymbol{y}}\right|$ 在 $\boldsymbol{y}_0$ 的值，代表「单位体积元由 $\boldsymbol{y}_0 \mapsto \boldsymbol{x}_0$ 的变化率」。（下面会看到，$\boldsymbol{F}$ 和 $\boldsymbol{F}^{-1}$ 的正/反方向不太重要，因为基本上不影响计算复杂度。）



(6)

每次得到**正奖励**，我们会令 Jacobian $|J|$ 增加一点：

$$|J| := \det \left[\frac{\partial \boldsymbol{F}^{-1}(\boldsymbol{y})}{\partial \boldsymbol{y}}\right]_{n\times n} \tag{7}$$

下标表示那是一个 $n \times n$ 矩阵。

其实 Jacobian 矩阵的意义就是：

$$J = \left[\frac{\partial \text{ 输出}}{\partial \text{ 输入}}\right] \tag{8}$$

神经网络的输入和输出都是 $\dim n$，所以 Jacobian 很自然是 $n \times n$ 矩阵。

用**梯度下降法**，我们需要计算这些梯度：$\left[\dfrac{\partial |J|}{\partial \boldsymbol{W}}\right]$，总数是网络中的 weights 的个数 $= \sum m_\ell$。

要用到 determinant 的微分公式：

$$\frac{d}{dt}|A(t)| = tr(\text{adj}(A) \cdot \frac{dA(t)}{dt}) \tag{9}$$

$$\text{adj}(A) := |A| \cdot A^{-1} \tag{10}$$

换句话说，对於每个权重 $w := \overset{\ell}{W}_{ij}$，我们要计算：

$$\frac{\partial}{\partial w}|J| = tr(|J| \cdot J^{-1} \cdot \left[\frac{\partial J}{\partial w}\right]) \tag{11}$$

注意：$|J|$ 和 J^{-1} 是 $\boldsymbol{y}_0$ 的函数，只需在大 loop 外一次过计算。

问题是，计算 $\left[\dfrac{\partial J}{\partial w}\right]_{n\times n}$ 的时候：

$$\frac{\partial J}{\partial w} = \frac{\partial}{\partial w}\frac{\partial \boldsymbol{F}^{-1}}{\partial y} = \frac{\partial}{\partial w}\frac{\partial}{\partial \boldsymbol{y}} \quad \overset{1}{W} \bigcirc \ldots \overset{\ell}{W} \bigcirc \ldots \overset{L}{W} \bigcirc \; \boldsymbol{y} \tag{12}$$

这牵涉到用 w 对 W^{-1} 的分量微分，可以想像就算计了出来也会是极复杂的。解决办法是，索性「**本末倒置**」，用 $\eqslantgtr$ 来定义神经网络，然后在 forward propagation 时才用 $W = \eqslantgtr^{-1}$ 计算。

J 的分量写出来是：

$$J_{ij} = \frac{\partial \boldsymbol{F}_i^{-1}}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\overset{1}{\eqslantgtr} \circledS ... \overset{\ell}{\eqslantgtr} \circledS ... \overset{L}{\eqslantgtr} \circledS \, \boldsymbol{y} \right]_i =: \nabla_{ij}^1 \tag{13}$$

$$\begin{cases} \nabla_{ij}^1 & := \sum\limits_{k_1} [\, \overset{1}{\eqslantgtr}_{ik_1} \circledS'(y_{k_1}^2) \nabla_{ij}^2 \,] \\ \nabla_{ij}^\ell & := \sum\limits_{k_\ell} [\, \overset{\ell}{\eqslantgtr}_{k_{\ell-1}k_\ell} \circledS'(y_{k_\ell}^{\ell+1}) \nabla_{ij}^{\ell+1} \,] \\ \nabla_{ij}^L & := \qquad \overset{L}{\eqslantgtr}_{k_{L-1}j} \circledS'(y_j) \end{cases} \tag{14}$$

这情况完全类似於经典 Back Prop，以上只是 chain rule 的应用，∇^ℓ 将每层用 chain rule 分拆开来，所以 ∇ 又叫 “local gradient”。上式就是整个网络的**反向传递**，其中每个 weight 出现 exactly 一次。

但工作还未完，我们要计算 $\dfrac{\partial J_{ij}}{\partial {\scriptstyle\eqslantgtr}} = \dot{\nabla}_{ij}^1$。（定义 ${\scriptstyle\eqslantgtr} := \overset{\ell}{\eqslantgtr}_{gh}$, $k_0 := i$, $k_L := j$）

注意：$\boldsymbol{x} = \boldsymbol{F}^{-1}(\boldsymbol{y})$，所以 $\boldsymbol{y}$ 是**自变量**，${\scriptstyle\eqslantgtr}$ 不影响 $\boldsymbol{y}$，所以 $\dfrac{\partial \boldsymbol{y}}{\partial {\scriptstyle\eqslantgtr}} \equiv 0$。

${\scriptstyle\eqslantgtr}$ 必会是 $\eqslantgtr$ 的其中一元，但如果 ${\scriptstyle\eqslantgtr} \notin \eqslantgtr$，以下的项微分后都会变成 0：

$$\begin{cases} \dot{\nabla}_{ij}^1 = \sum\limits_{k_1} \left[\overset{1}{\eqslantgtr}_{ik_1} \circledS'(y_{k_1}^2) \dot{\nabla}_{ij}^2 \right] \\ \dot{\nabla}_{ij}^\ell = \sum\limits_{k_\ell} \left[\overset{\ell}{\eqslantgtr}_{k_{\ell-1}k_\ell} \circledS'(y_{k_\ell}^{\ell+1}) \dot{\nabla}_{ij}^{\ell+1} \right] \\ \dot{\nabla}_{ij}^L = \qquad \overset{L}{\eqslantgtr}_{k_{L-1}j} \circledS'(y_j) \equiv 0 \end{cases} \tag{15}$$

所以实际上只剩下一项：

$$\frac{\partial J_{ij}}{\partial {\scriptstyle\eqslantgtr}} = \sum_{k_1} \left[\overset{1}{\eqslantgtr}_{k_0k_1} \circledS'(y_{k_1}^2) ... \sum_{k_{\ell-2}} \left[\overset{\ell-2}{\eqslantgtr}_{k_{\ell-3}k_{\ell-2}} \circledS'(y_{k_{\ell-2}}^{\ell-1}) ... \right.\right. \tag{16}$$

$$\begin{cases} ... \quad \overset{\ell-1}{\eqslantgtr}_{k_{\ell-2},g} \circledS'(y_g^\ell) \circledS'(y_h^{\ell+1}) \nabla_{ij}^{\ell+1} \Big]\Big] \\ ... \quad \overset{\ell-1}{\eqslantgtr}_{k_{\ell-2},g} \circledS'(y_g^\ell) \circledS'(y_h^{\ell+1}) \Big]\Big] \qquad \text{if } {\scriptstyle\eqslantgtr} \in \text{last layer} \end{cases}$$

上式的意思是：每层 layer 重复一块 $[\sum \eqslantgtr \circledS]$，直到遇到 ${\scriptstyle\eqslantgtr} = \overset{\ell}{\eqslantgtr}_{gh}$，则用结尾形式取代之。

和经典 Back Prop 不同的是，上式只是 $n \times n$ 矩阵中的一个元素，从复杂度而论，每个 weight 的 ∇ 计算，增加了起码 n^2 倍的复杂度（虽然其计算上可以共用一些结果）。记住 $n = \dim \boxed{\text{状态空间}}$。

可以这样理解：每个 weight 的调教，需要计算这个 weight 对 Jacobian 的影响，而那 Jacobian 是**整个**网络的特性。关键似乎就在於每个 weight 对 Jacobian 的**影响**。

现在回看更高层次的这个式子：

$$
\begin{aligned}
\frac{\partial}{\partial w}|J| &= tr(|J| \cdot J^{-1} \cdot \left[\frac{\partial J}{\partial w}\right]) && (17)\\
&= |J| \cdot tr(\left[\frac{\partial y}{\partial x}\right] \cdot \left[\frac{\partial}{\partial w}\frac{\partial x}{\partial y}\right]) && (18)\\
&= |J| \cdot \sum_{ij} \left(\frac{\partial y}{\partial x}\right)_{ij} \left(\frac{\partial}{\partial w}\frac{\partial x}{\partial y}\right)_{ij} && (19)
\end{aligned}
$$

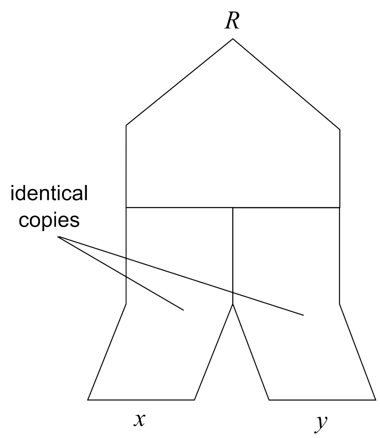
上式中最重要（最慢）的是那 $(i,j) \in n \times n$ 求和。裡面的第一個因子是 Jacobian J，第二个因子是我们刚计算了的 $\nabla_w J^{-1}$。

Back Prop 的 ∇ 形式上是 $\dfrac{\partial\ \text{输出}}{\partial\ \mathrm{w}}$，我们的 ∇ 形式是 $\left[\dfrac{\partial}{\partial w}\dfrac{\partial\ \text{输入}}{\partial\ \text{输出}}\right]_{n\times n}$。

其实我们只需要计算 $\nabla_w|J|$ 的**大约方向**。暂时我在代码中的做法是：忽略式 (19) 中较小的项，那就不需做足 n^2 个乘积。

或者可不可以将 $|J|(\geqslant)$ 看成是一个 weight $\geqslant$ 的函数，然后用它的 Taylor series expansion 来近似？

Notes



(20)

或者，照旧是 feedforward network，但 somehow 它的学习是基於某 reward function。问题是这 reward function 从何而来？R 可以是一个全部 W 的函数，是由我们任意定义的。

1 Jacobian learning algorithm

The classical Back Prop algorithm is **error-driven**, but in many AI problems the "correct" answers are not given, instead the feedback is provided via **rewards**. When an answer is correct, the reward $R > 0$, error $\mathscr{E} = 0$; when the answer is incorrect, $R < 0$, but $\mathscr{E}$ is still unknown (because we don't know the correct answer). In other words, error-driven learning is inapplicable.

So I thought of a reward-driven learning method: assume the neural network maps $\boldsymbol{x}_0 \mapsto \boldsymbol{y}_0$, usually it also maps the neighborhood of $\boldsymbol{x}_0$ to the neighborhood of $\boldsymbol{y}_0$. If we wish to "strengthen" this pair of mapping, we can make a **bigger** neighborhood of $\boldsymbol{x}_0$ map to the same neighborhood close to $\boldsymbol{y}_0$. We will make this precise.

This type of learning algorithm should be very useful to AI, and currently the author is not aware of other alternatives for training [deep] neural networks via rewards.

A feed-forward neural network can be constructed this way:

$$\boldsymbol{y} = \boldsymbol{F}(\boldsymbol{x}) \tag{21}$$

$$\boldsymbol{y} = \oslash \overset{L}{W} \ldots \oslash \overset{\ell}{W} \ldots \oslash \overset{1}{W} \boldsymbol{x} \tag{22}$$

where W represents the matrix of **weights** on each layer ℓ.

The **inverse** of F is:

$$\boldsymbol{x} = \boldsymbol{F}^{-1}(\boldsymbol{y}) \tag{23}$$

$$\boldsymbol{x} = \overset{1}{\gtreqless} \obslash \ldots \overset{\ell}{\gtreqless} \obslash \ldots \overset{L}{\gtreqless} \obslash \, \boldsymbol{y} \tag{24}$$

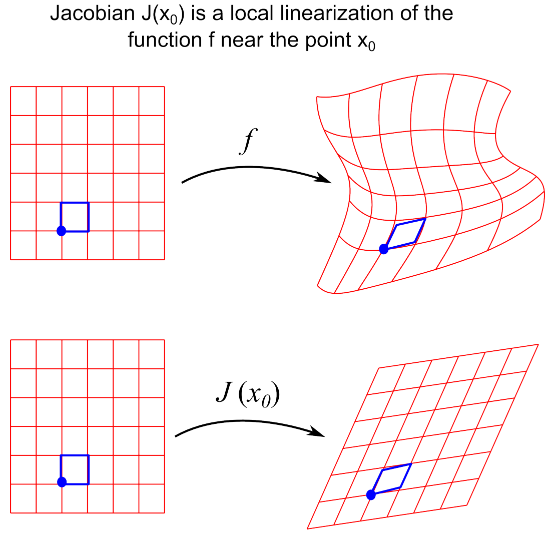
Note: $\gtreqless = W^{-1}$, $\obslash = \oslash^{-1}$, the shape is different.

Assume that in the space of $\boldsymbol{x}$ there is a volume element U, which transforms via $\boldsymbol{F}$ to a volume element V in the space of $\boldsymbol{y}$, then:

$$U = |J| \cdot V \tag{25}$$

$J = \left[\dfrac{\partial \boldsymbol{F}(x)}{\partial x} \right]$ is called the **Jacobian** matrix.

In our case, the value of $|J| = \left| \dfrac{\partial \boldsymbol{F}^{-1}(\boldsymbol{y})}{\partial \boldsymbol{y}} \right|$ at $\boldsymbol{y}_0$ represents "the change in volume from $\boldsymbol{y}_0 \mapsto \boldsymbol{x}_0$". (Below, we will see that the direction of $\boldsymbol{F}$ or $\boldsymbol{F}^{-1}$ is not too important, as either way the computational complexity is essentially the same.)



(26)

Every time we get a **positive reward**, we can let the Jacobian $|J|$ increase slightly:

$$|J| := \det \left[\frac{\partial \boldsymbol{F}^{-1}(\boldsymbol{y})}{\partial \boldsymbol{y}} \right]_{n \times n} \tag{27}$$

The subscript indicates that it is a $n \times n$ matrix.

The meaning of the Jacobian matrix is:

$$J = \left[\frac{\partial \text{ input}}{\partial \text{ output}} \right] \tag{28}$$

The neural network's input and output are both of $\dim n$, so the Jacobian is naturally an $n \times n$ matrix.

To use **gradient descent**, we need to calculate these gradients: $\left[\frac{\partial |J|}{\partial \boldsymbol{W}} \right]$, their total number is the number of weights in the network $= \sum \ell \, \#(\overset{\ell}{W})$.

We need the formula for the derivative of the determinant:

$$\frac{d}{dt}|A(t)| = tr(\text{adj}(A) \cdot \frac{dA(t)}{dt}) \tag{29}$$

$$\text{adj}(A) := |A| \cdot A^{-1} \tag{30}$$

In other words, for each weight $w := \overset{\ell}{W}_{ij}$, we need to calculate:

$$\frac{\partial}{\partial w}|J| = tr(|J| \cdot J^{-1} \cdot \left[\frac{\partial J}{\partial w} \right]) \tag{31}$$

Note: $|J|$ and J^{-1} are functions of $\boldsymbol{y}_0$, we only need to calculate them once outside the big loop.

Now a problem arises in the calculation of $\left[\frac{\partial J}{\partial w} \right]_{n \times n}$:

$$\frac{\partial J}{\partial w} = \frac{\partial}{\partial w} \frac{\partial \boldsymbol{F}^{-1}}{\partial y} = \frac{\partial}{\partial w} \frac{\partial}{\partial \boldsymbol{y}} \; \overset{1}{W} \bigcirc \ldots \overset{\ell}{W} \bigcirc \ldots \overset{L}{W} \bigcirc \, \boldsymbol{y} \tag{32}$$

This requires us to differentiation W^{-1} w.r.t. w; we can imagine the result would be very complicated. So we use a trick, by "reversing" the network, we use $\eqslantgtr$ to define the network weights, and then use $W = \eqslantgtr^{-1}$ during forward propagation.

The components of J are:

$$J_{ij} = \frac{\partial \boldsymbol{F}_i^{-1}}{\partial y_j} = \frac{\partial}{\partial y_j} \left[\overset{1}{\eqslantgtr} \oslash ... \overset{\ell}{\eqslantgtr} \oslash ... \overset{L}{\eqslantgtr} \oslash \boldsymbol{y} \right]_i =: \nabla_{ij}^1 \tag{33}$$

$$\begin{cases} \nabla_{ij}^1 & := \sum_{k_1} [\overset{1}{\eqslantgtr}_{ik_1} \oslash'(y_j^2) \nabla_{ij}^2] \\ \nabla_{ij}^\ell & := \sum_{k_\ell} [\overset{\ell}{\eqslantgtr}_{k_{\ell-1}k_\ell} \oslash'(y_j^{\ell+1}) \nabla_{ij}^{\ell+1}] \\ \nabla_{ij}^L & := \quad \overset{L}{\eqslantgtr}_{k_{L-1}j} \oslash'(y_j) \end{cases} \tag{34}$$

This situation is exactly analogous to the classical Back Prop algorithm; The above is just the application of the **chain rule**, with ∇^ℓ written separately for each layer, therefore ∇ is called the "local gradient". The above formula amounts to propagating the entire network one time, where every weight appears **exactly once**.

But our work is not finished yet; We need to calculate $\dfrac{\partial J_{ij}}{\partial {\scriptstyle\eqslantgtr}} =: \dot{\nabla}_{ij}^1$.

(Let's define ${\scriptstyle\eqslantgtr} := \overset{\ell}{\eqslantgtr}_{gh}$, $k_0 := i$, $k_L := j$)
${\scriptstyle\eqslantgtr}$ would be an element of $\eqslantgtr$, but if ${\scriptstyle\eqslantgtr} \notin \eqslantgtr$, all the terms below would vanish:

$$\begin{cases} \dot{\nabla}_{ij}^1 = \sum_{k_1} \left[\overset{1}{\eqslantgtr}_{ik_1} \oslash'(y_j^2) \dot{\nabla}_{ij}^2 \right] \\ \dot{\nabla}_{ij}^\ell = \sum_{k_\ell} \left[\overset{\ell}{\eqslantgtr}_{k_{\ell-1}k_\ell} \oslash'(y_j^{\ell+1}) \dot{\nabla}_{ij}^{\ell+1} \right] \\ \dot{\nabla}_{ij}^L = \quad \overset{L}{\eqslantgtr}_{k_{L-1}j} \oslash'(y_j) \equiv 0 \end{cases} \tag{35}$$

So what is left over is just this term:

$$\frac{\partial J_{ij}}{\partial {\scriptstyle\eqslantgtr}} = \sum_{k_1} \left[\overset{1}{\eqslantgtr}_{k_0k_1} \oslash'(y_j^2) ... \sum_{k_\ell} \left[\overset{\ell}{\eqslantgtr}_{k_{\ell-1}k_\ell} \oslash'(y_j^{\ell+1}) ... \right.\right. \tag{36}$$
$$\begin{cases} ... \ \oslash'(y_j^{\ell+1}) \nabla_{ij}^{\ell+1}]] & \\ ... \ \oslash'(y_j^{\ell+1})]] & \text{if } {\scriptstyle\eqslantgtr} \in \text{last layer} \end{cases}$$

The above formula means: For each layer we repeat a block of $[\sum \eqslantgtr \oslash]$, until we encounter ${\scriptstyle\eqslantgtr} = \overset{\ell}{\eqslantgtr}_{gh}$, then we replace with the terminal form.

In contrast to classical Back Prop, the above formula gives us only one element in an $n \times n$ matrix; From the complexity point of view, calculating the ∇ for each weight is at least n^2 times as costly as classical Back Prop (even though we may re-use some intermediate computation results). Recall that $n = \dim \boxed{\text{state space}}$.

We can understand it thusly: For each weight we try to calculate its influence towards the Jacobian, but the Jacobian is a **global** property of the network. The key seems to lie in how each weight **influences** the Jacobian.

Now let's look back at this higher-level formula:

$$
\begin{aligned}
\frac{\partial}{\partial w}|J| &= tr(|J| \cdot J^{-1} \cdot \left[\frac{\partial J}{\partial w}\right]) && (37)\\
&= |J| \cdot tr(\left[\frac{\partial y}{\partial x}\right] \cdot \left[\frac{\partial}{\partial w}\frac{\partial x}{\partial y}\right]) && (38)\\
&= |J| \cdot \sum_{ij} \left(\frac{\partial y}{\partial x}\right)_{ij} \left(\frac{\partial}{\partial w}\frac{\partial x}{\partial y}\right)_{ij} && (39)
\end{aligned}
$$

The most critical (slowest) part is the $(i, j) \in n \times n$ summation. The first factor inside $\sum$ is the Jacobian J, the second factor is the $\nabla_w J^{-1}$ that we just calculated.

Back Prop's ∇ has the form $\dfrac{\partial \boxed{\text{output}}}{\partial \boxed{\text{weights}}}$

whereas our ∇ has the form $\left[\dfrac{\partial}{\partial \boxed{\text{weights}}} \dfrac{\partial \boxed{\text{input}}}{\partial \boxed{\text{output}}}\right]_{n\times n}$.

In fact we just need to calculate the **approximate** direction and size of $\nabla_w|J|$. Currently in our code we use this trick: ignore the smaller terms in (39), so we don't need to do all of n^2 products.

Or perhaps we can regard $|J|(\eqslantgtr)$ as a function of the weight $\eqslantgtr$, and then use its Taylor series expansion to approximate?